# DL-HW #04

2015004693_양상헌

*(실행환경: Jupyter Notebook)*

## 1. Source Code:

In [4]:

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)

X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# ====Assignment=================================

#  784 -> 256 -> 128 -> 256 -> 10     (total 4-layer Design)

W1 = tf.Variable(tf.random_uniform([784, 256], -1., 1.))
b1 = tf.Variable(tf.random_uniform([256], -1., 1.))
L1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_uniform([256, 128], -1., 1.))
b2 = tf.Variable(tf.random_uniform([128], -1., 1.))
L2 = tf.sigmoid(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_uniform([128, 256], -1., 1.))
b3 = tf.Variable(tf.random_uniform([256], -1., 1.))
L3 = tf.sigmoid(tf.matmul(L2, W3) + b3)

W4 = tf.Variable(tf.random_uniform([256, 10], -1., 1.))
b4 = tf.Variable(tf.random_uniform([10], -1., 1.))
logits = tf.matmul(L3, W4) + b4
hypothesis = tf.nn.softmax(logits)

# ===============================================
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels = Y, logits = logits))
opt = tf.train.GradientDescentOptimizer(learning_rate=1.0).minimize(cost)
batch_size = 100

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for epoch in range(15):
        avg_cost = 0
        total_batch = int(mnist.train.num_examples/batch_size)

        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            c, _ = sess.run([cost, opt], feed_dict={X:batch_xs, Y: batch_ys})
            avg_cost += c / total_batch
        print('Epoch:', '%d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

    is_correct = tf.equal(tf.argmax(hypothesis,1),tf.argmax(Y,1))
    accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
    print("Accuracy", sess.run(accuracy, feed_dict={X:mnist.test.images, Y:mnist.test.labels}))
```

## 2. Result:

```
Extracting ./mnist/data/train-images-idx3-ubyte.gz
Extracting ./mnist/data/train-labels-idx1-ubyte.gz
Extracting ./mnist/data/t10k-images-idx3-ubyte.gz
Extracting ./mnist/data/t10k-labels-idx1-ubyte.gz
Epoch: 1 cost = 2.022304504
Epoch: 2 cost = 0.220650661
Epoch: 3 cost = 0.154509364
Epoch: 4 cost = 0.122302119
Epoch: 5 cost = 0.096122807
Epoch: 6 cost = 0.078478723
Epoch: 7 cost = 0.063671677
Epoch: 8 cost = 0.050665227
Epoch: 9 cost = 0.040454531
Epoch: 10 cost = 0.032597325
Epoch: 11 cost = 0.025815802
Epoch: 12 cost = 0.020912775
Epoch: 13 cost = 0.016774750
Epoch: 14 cost = 0.013472038
Epoch: 15 cost = 0.010779920
Accuracy 0.9658
```

## 3. Disscussion:

< 코드 설명 >

코드의 전체적인 틀은 실습 자료 "3.MLP(2).pdf"에 나와있는 Skeleton Code를 그대로 가져와 사용하였고, GradientDescentOptimizer()의 Learning_rate만 기존의 0.1에서 1로 바꾸어 더 빠른 속도로 학습이 이루어질 수 있도록 하였다. 전체 트레이닝 데이터를 총 15번 학습하게 되고, 전체 트레이닝 데이터가 1번 학습될 때의 batch의 사이즈는 100으로 하여 전체 데이터 숫자인 55000 에 100을 나눈 결과인 550만큼 for문을 반복하며 학습하게 된다. 또한 15번 학습을 반복할 때, 전체 트레이닝 데이터에 대한 평균 cost를 출력하도록 하고, 마지막에 최종 test 데이터로 모델의 정확성을 확인하여 이를 수치상으로 출력하도록 해주었다.

기본적인 MLP의 구조는 코드 상에 언급되어 있는대로 각각 784->256->128->256->10개의 뉴런으로 구성되고 총 4개의 Layer로 이뤄져있는 네트워크의 모델로 디자인 하였다. 이때 마지막 Layer를 제외한 나머지 Layer에서는 Activation Function으로 Sigmoid를 사용하고, 마지막 Layer에서는 총 10개의 Label로 분류해야하는 작업이므로, Sigmoid가 아닌 Softmax를 Activation Function으로 사용하였다.

< 결과 분석 >

95%의 정확도를 목표로 하여 디자인 하였고 출력된 결과는 0.9658로써 약 96.6%의 정확도를 나타내고 목표한 정확도에 잘 도달한 것을 확인할 수 있다. 또한 같은 트레이닝 데이터를 계속해서 반복하여 학습 시킬 때마다 점점 평균 Cost가 줄어드는 것을 확인할 수 있다.